Технології програмування

Лабораторна робота №4

**Зворотний польський запис та обчислення математичного виразу**

**Мета роботи**: Використовуючи теоретичне підґрунтя про зворотний польський запис розробити програму на вхід якої подається математичний вираз, що має довільний набір операндів, операторів та дужок, на виході програма обчислює результат математичного виразу.

**Теоретичні відомості**

**Зворотний польський запис**

Звичною формою запису виразів є інфіксна, коли знак бінарної операції записують між позначеннями операндів цієї операції, наприклад, a + b. Розглянемо запис знаків операцій після позначень операндів, тобто постфіксний запис, наприклад, a b +. Такий запис має також назву зворотного польського, бо його запропонував польський логік Ян Лукасевич. Далі словосполучення: «зворотний польський запис» позначатимемо ЗПЗ. Позначення для функції традиційно записують перед аргументами. Природно такий запис назвати префіксним. При описі ЗПЗ переважно обмежуються перетворенням інфіксного запису у ЗПЗ.

Зворотний польський запис має чудові властивості, які перетворюють її на ідеальну проміжну ланку при трансляції коду програми.

Обчислення виразу, записаного в зворотному польському записі, можна проводити шляхом однократного перегляду ЗПЗ.

Зворотний польський запис виразу з арифметичними діями та піднесенням до степеня можна отримати, дотримуючись алгоритму, запропонованого Дейкстpою. Алгоритм отримав назву «сортувальна станція», за подібність його операцій із тим, що відбувається на залізничних сортувальних станціях. Як і алгоритм обчислення ЗПЗ, алгоритм сортувальної станції ґрунтується на стеку. У перетворенні беруть участь дві текстові змінні: вхідний і вихідний рядки. У процесі перетворення використовується стек, що зберігає ще не додані до вихідного рядка операції. Перетворювальна програма читає вхідний рядок послідовно символ за символом (символ – це не обов'язково буква), виконує на кожному кроці деякі дії залежно від того, який символ було прочитано.

**Алгоритм**

У випадку, коли є символи для обробки необхідно зчитати символ.

Якщо символ є числом або додаємо його до вихідного рядка.

Якщо символ є дужкою, поміщаємо його в стек.

Якщо символ є дужкою, що закривається то доки верхнім елементом стека не стане відкриваюча дужка, виштовхуємо елементи зі стека у вихідний рядок. При цьому дужка, що відкриває, видаляється зі стека, але у вихідний рядок не додається. Якщо стек закінчився раніше, ніж ми зустріли дужку, це означає, що у виразі або неправильно поставлений роздільник, або не узгоджені дужки.

Якщо символ є бінарною операцією та операція на вершині стеку має більший або такий самий пріоритет, то необхідно “виштовхнути” верхній елемент до вихідного рядка. Помістити операцію в стек.

Коли вхідний рядок закінчився, виштовхуємо всі символи зі стека у вихідний рядок

**Пріоритетність операцій:**

Найвищий – вираз в дужках.

Високий – піднесення до степеня.

Середній – множення або ділення.

Низький – додавання або віднімання.

**Приклад:**

|  |
| --- |
| Вхід: 3 + 4 \* 2 / (1 - 5) ^ 2  Читаємо «3»  Додаємо «3» до вихідного рядка  Вихід: 3  Читаємо «+»  Кладемо «+» у стек  Вихід: 3  Стек: +  Читаємо «4»  Додамо «4» до вихідного рядка  Вихід: 3 4  Стек: +  Читаємо «\*»  Кладемо «\*» у стек  Вихід: 3 4  Стек: + \*  Читаємо «2»  Додамо «2» до вихідного рядка  Вихід: 3 4 2  Стек: + \*  Читаємо «/»  Виштовхуємо «\*» зі стека у вихідний рядок, кладемо «/» у стек  Вихід: 3 4 2 \*  Стек: +/  Читаємо «(»  Кладемо «(» у стек  Вихід: 3 4 2 \*  Стек: + / (  Читаємо «1»  Додамо «1» до вихідного рядка  Вихід: 3 4 2 \* 1  Стек: + / (  Читаємо «−»  Кладемо «−» у стек  Вихід: 3 4 2 \* 1  Стек: + / ( −  Читаємо «5»  Додамо «5» до вихідного рядка  Вихід: 3 4 2 \* 1 5  Стек: + / (-  Читаємо «)»  Виштовхуємо «−» зі стека у вихідний рядок, виштовхуємо «(»  Вихід: 3 4 2 \* 1 5 −  Стек: +/  Читаємо «^»  Кладемо «^» у стек  Вихід: 3 4 2 \* 1 5 −  Стек: +/^  Читаємо «2»  Додамо «2» до вихідного рядка  Вихід: 3 4 2 \* 1 5 − 2  Стек: +/^  Кінець виразу  Виштовхуємо всі елементи зі стека в рядок  Вихід: **3 4 2 \* 1 5 − 2 ^ / +** |

**Обчислення виразу**

Використовуючи алгоритм ЗПЗ математичний вираз 3 + 4 \* 2 / (1 - 5) ^ 2 був записаний у вигляді 3 4 2 \* 1 5 − 2 ^ / +

Обчислення проводиться зліва направо. Якщо в запису зустрічається число, то число поміщається в стек. Якщо в запису зустрічається оператор, то він застосовується до двох верхніх елементів стеку які виштовхуються із стеку, а результат виконання поміщається в стек.

Запис інтерпретується як зазначено у наведеній нижче таблиці (зазначено стан стека після виконання операції, вершина стека виділена червоним кольором)

|  |  |  |
| --- | --- | --- |
| Символ | Дія | Стек |
| 3 | помістити в стек | 3 |
| 4 | помістити в стек | 3 4 |
| 2 | помістити в стек | 3 4 2 |
| \* | множення | 3 8 |
| 1 | помістити в стек | 3 8 1 |
| 5 | помістити в стек | 3 8 1 5 |
| - | віднімання | 3 8 -4 |
| 2 | помістити в стек | 3 8 -4 2 |
| ^ | піднесення до степеню | 3 8 16 |
| / | ділення | 3 0.5 |
| + | додавання | 3.5 |

Результат **3.5**, в кінці обчислень знаходиться на вершині стека.

**Хід виконання завдання до лабораторної роботи**

1. Прочитав суть завдання.

2. Відкрив графічний редактор.

3. Написав код, який виконує завдання.

4. Написав звіт про виконання завдання.

5. Створив і виконав Юніт тести.

6. Завантажив всі зміни для розгляду в гілку Labs.

Код файлу lab\_04.py:

|  |
| --- |
| import operator  from collections import deque  class ExpressionEvaluator:  def \_\_init\_\_(self):  self.operators = {'+': operator.add, '-': operator.sub, '\*': operator.mul, '/': operator.truediv, '^': operator.pow}  def is\_numeric(self, unit):  return unit.replace('.', '', 1).isdigit()  def get\_operator\_priority(self, operator):  priority = {'+': 0, '-': 0, '\*': 1, '/': 1, '^': 2}  return priority.get(operator, -1)  def to\_rpn(self, input\_str):  stack = []  output\_str = []  for token in input\_str.split(' '):  if self.is\_numeric(token):  output\_str.append(token)  continue  if token == '(':  stack.append(token)  continue  if token == ')':  for op in reversed(stack):  if op == '(':  stack.remove(op)  break  output\_str.append(op)  stack.remove(op)  continue  if token in self.operators:  op\_power = self.get\_operator\_priority(token)  if len(stack) >= 1 and stack[-1] in self.operators:  last\_op\_in\_stack\_power = self.get\_operator\_priority(stack[-1])  if last\_op\_in\_stack\_power >= op\_power:  output\_str.append(stack[-1])  stack.remove(stack[-1])  stack.append(token)  output\_str.extend(reversed(stack))  return output\_str  def evaluate\_rpn(self, rpn\_expression):  stack = []  for token in rpn\_expression:  if self.is\_numeric(token):  stack.append(token)  else:  num2 = float(stack[-1])  num1 = float(stack[-2])  operator\_function = self.operators[token]  result = operator\_function(num1, num2)  del stack[-1]  del stack[-1]  stack.append(result)  return stack  if \_\_name\_\_ == '\_\_main\_\_':  evaluator = ExpressionEvaluator()  user\_input = input("Enter your expression: ")  print(f"User's expression: {user\_input}")  try:  rpn\_result = evaluator.to\_rpn(user\_input)  print(f"Reverse Polish Notation: {' '.join(map(str, rpn\_result))}")  result = evaluator.evaluate\_rpn(rpn\_result)  print(f"Result: {result}")  except ValueError as e:  print(f"Error: {e}") |

Файл із Юніт тесами:

|  |
| --- |
| import pytest  from lab\_04 import ExpressionEvaluator  @pytest.fixture  def evaluator():  return ExpressionEvaluator()  def test\_numeric\_check(evaluator):  assert evaluator.is\_numeric("123")  assert evaluator.is\_numeric("3.14")  assert not evaluator.is\_numeric("+")  assert not evaluator.is\_numeric("(")  def test\_operator\_priority(evaluator):  assert evaluator.get\_operator\_priority('+') == 0  assert evaluator.get\_operator\_priority('-') == 0  assert evaluator.get\_operator\_priority('\*') == 1  assert evaluator.get\_operator\_priority('/') == 1  assert evaluator.get\_operator\_priority('^') == 2  assert evaluator.get\_operator\_priority('%') == -1  def test\_to\_rpn(evaluator):  input\_str = "( 18 \* 2 ) + ( 64 / 8 ) ^ 4"  expected\_rpn = ['18', '2', '\*', '64', '8', '/', '4', '^', '+']  assert evaluator.to\_rpn(input\_str) == expected\_rpn  def test\_evaluate\_rpn(evaluator):  rpn\_expression = ['18', '2', '\*', '64', '8', '/', '4', '^', '+']  assert evaluator.evaluate\_rpn(rpn\_expression) == [4132.0] |